AD-A089 461 VIRGINIA POLYTECHNIC INST AND STATE UNIV BLACKSBURG --ETC F/G 9/2
GENERIC DATA TRANSACTION SYSTEM. VERSION 2. SYSTEM DESIGN.(U)
MAR 79 J E EVANS, L H MASON, R C WILLIGES N00123-78-C-1401
UNCLASSIFIED VPI-HFL-79-2/NPRDC-79-1 NL

END
DATE FILMED
10-80
DTIC

(2)

GENERIC DATA TRANSACTION SYSTEM:

VERSION 2 SYSTEM DESIGN

by

John E. Evans, III

L. Hardy Mason

HFL-79-2/NPRDC-79-1

March 1979

Prepared Under Contract to

Navy Personnel Research and Development Center

San Diego, California 92152

Contract No. N00123-78-C-1401

Principal Investigator

Robert C. Williges

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

(9)

4. TITLE (and Subtitle)

(6) GENERIC DATA TRANSACTION SYSTEM, VERSION 2, SYSTEM DESIGN. (12)

VPI

5. TYPE OF REPORT & PERIOD COVERED

Technical Report
15 July 1978 - 15 July 1979

6. PERFORMING ORG. REPORT NUMBER

HFL-79-2/NPRDC-79-1

7. AUTHOR(s)

(10) John E. Evans, III
L. Hardy Mason
Robert C. Williges (5)

8. CONTRACT OR GRANT NUMBER(s)

N00123-78-C-1401
N00123-77-C-1071

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Human Factors Laboratory
Dept. of Industrial Engineering & Operations Res.
Virginia Polytechnic Institute & State University
Blacksburg, VA 24061

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

62757N (HF & Simulation Technology)
ZF57-525-001-022 (HF: Manned Systems Design)

11. CONTROLLING OFFICE NAME AND ADDRESS

(11) Design of Manned Systems Program
Navy Personnel Research & Development Center
San Diego, California 92152

12. REPORT DATE

March 1979

(12) 184

13. NUMBER OF PAGES

74

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

(16) F57525
(17) ZF57525001

15. SECURITY CLASS. (of this report)

Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

plasma panel application, minicomputer applications, information base software, transaction system, table driven software, touch panel applications, event based emulator, embedded operator performance measurement, computer aided behavior research

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report and its companion (HFL-79-3/NPRDC-79-2) supercede HFL-78-2/NPRDC-78-2, documenting an enhanced version of the same system. The program described is a machine dependent, table driven, general software system written for the advanced programmer to communicate the internal operations of the software. Included are a description of data structures, program logic, and user interfaces. The Fortran IV software processes randomly accessed data which are tree structured by file, record, page, and field, and includes a functionally complete primitive command language, a display format processor, and provision for user supplied software appendages.

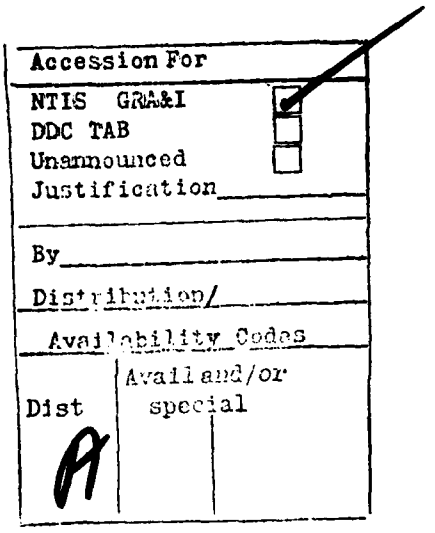
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

411032

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

This report and its companion (HFL-79-3/NPRDC-79-2) supercede HFL-78-2/NPRDC-78-2, a previous report by the same authors. (Evans, J.E., III and Mason, L.H. Generic data transaction system: system design. Blacksburg, Virginia: Virginia Polytechnic Institute and State University, HFL-78-2/NPRDC-78-2, July, 1978.)



ACKNOWLEDGMENT

This report describes work performed at Virginia Polytechnic Institute and State University under contracts N00123-77-C-1081 and N00123-78-C-1401 with the Navy Personnel Research and Development Center. Dr. Frederick Muckler served as the scientific monitor. The opinions expressed in this report are those of the authors, and do not necessarily represent the views of the Navy Personnel Research and Development Center or the U.S. Navy.

SUMMARY

Problem

A software system is required which will allow the experimental investigation of embedded operator performance evaluation techniques in data transaction systems. To permit meaningful research, the system must provide the experimenter a wide latitude in the design and generation of a data transaction task.

Approach

A generic data transaction system was appraised to be a suitable vehicle for attaining the above stated goals. Such a system would consist of a functionally complete set of primitive operators which could be used to process a random access dataset. Through the use of a hierarchically organized description, a programmer would communicate to the generic system the definition of a specific application, i.e., record and display formats. The additional provision for user supplied software appendages at strategic points in the generic system imparts syntactic and semantic extensibility to the native minimal query language.

Results

A generic data transaction system was implemented in FORTRAN IV and Assembly language on a minicomputer with 28K words of memory, a cartridge disk subsystem, a programmable real time clock, and a plasma panel terminal (including 512 by 512 pixel ac plasma panel, 32 by 32 resolution touch sensitive panel, and ASCII or PLATO keyset). The flexibility and extensibility of the generic system were found to be acceptable for the implementation of experimental (research and non-research) applications.

Conclusion

A table driven generic data transaction system consisting of a functionally complete set of primitive operators and an extensible minimal query language is a desirable vehicle for allowing experimental investigation into many aspects of the design and utilization of data transaction systems. Additionally, due to the relative ease with which an application may be created or modified, the generic system has merit in a non-research environment as well.

Recommendation

The generic system approach which was applied to event-based systems in this investigation should also be applied to time-based systems with the expectation of similar encouraging results.

DISCLAIMER

The software system described in this document is the product of the authors alone and does not incorporate any suggestions from any other individuals or uncited sources. The authors make no guarantee of the accuracy of any part of this document or the software system and assume no responsibility or liability for any consequences of its usage under any conditions. Duplication of the documentation or software whether in part or in full by whatever means and for whatever purpose is permitted only with the inclusion of proper reference to authorship.

TABLE OF CONTENTS

I. INTRODUCTION

In order to allow the experimental investigation of embedded operator performance evaluation techniques in data transaction systems, a generic data transaction system was developed with which many different transaction systems could be implemented and studied. The subject interface to the generic system consists essentially of a 512 by 512 pixel ac plasma panel output device on which formatted alphanumeric and graphical information may be displayed, and a (ASCII or PLATO) keyboard and 32 by 32 resolution touch sensitive panel through which the subject may interact with the system.

To provide the flexibility necessary for the implementation by the experimenter of both novel and conventional but dissimilar data transaction systems, it was necessary that a table driven scheme be used in the design of the generic system. Through the use of the available tables (or descriptors) explained in this document, the experimenter is allowed a wide latitude in the design and generation of a data transaction task for use in his research. Also, the generic system provides a functionally complete set of primitive operations from which complex query languages may be constructed by the experimenter. Alternatively, if the experimenter desires, the primitive command language can be used directly as provided. The information base for the generic system consists of nonhomogeneous hierarchically organized (file,

record, page, field) binary coded data stored on direct access online mass storage.

The entire generic system was implemented on a PDP 11/55 minicomputer with 28K words of core memory, an RK05 disk drive with RK11 controller, a KW11-P real time clock, all manufactured by Digital Equipment Corp., and a GCC-1B plasma panel terminal manufactured by Information Technology Ltd.

This technical report is the first of three documents describing the Generic Data Transaction System. The second, (HFL-79-3/NPRDC-79-2), contains a complete set of source listings for the system. The third, (HFL-79-4/NPRDC-79-3), contains a user's guide and sample application-specific user routines.

II. CONTROL CYCLE

The execution of the generic data transaction system consists of an initialization phase during which files are opened, control variables are initialized, and completion routines are specified, followed by the execution phase which consists of a single loop within which either a command is interpreted and executed or a field update is applied. Asynchronously, the touch panel may be used in the specification of a field for updating. The optional user timing of character writing delays as a synchronous extension of the plasma panel interrupt handler occurs asynchronous to the rest of the execution phase and program termination occurs synchronously in response to a <CTRL-C> equivalent on the keyboard. The following outline summarizes the (optional) operations.

I. Initialization
   - A. Open files
   - B. Control variable assignments
   - C. Specify completion routines
   - (D. User supplied initialization)

II. Execution
   - A. User-performance recording
   - B. Input prompting (user supplied prompting)

C. Command input; also, translate end-of-file to <CTRL-Z> command, and error to <CTRL-C> command

D. User-performance recording

E. Update mode:

1. Preliminary verification or update request cancellation

2. (User supplied conversion (and field updating)) and/or conversion and field updating

3. Error reporting (user supplied messages) and user-performance recording

F. or command mode:

1. (User supplied command interpretation (and execution)) and/or command interpretation, and user-performance recording

2. Command execution - select from:

a. LOGON (user supplied messages)

i. Control record processing

ii. Device activation

(iii. User supplied extensions)

b. LOGOFF

i. Final record processing

ii. Control record processing

iii. Device deactivation

(iv. User supplied extensions)

c. SELECT (user supplied comparisons and/or messages)

d. RELEASE (user supplied messages)

e. ADD/INSERT

f. DELETE

g. Record accessing (user supplied messages)

h. Page accessing (user supplied timing)

i. Field accessing and switch to update mode

j. Update cancellation

3. Error reporting (user supplied messages) and user-performance recording

III. Touch panel (asynchronous)

A. Coordinate translation (user supplied preprocessing)

B. Field selection

C. Field accessing (user supplied timing) and switch to update mode

(IV. Character writing (user supplied timing) (asynchronous))

V. Termination

A. Drain I/O queues

B. Device termination

(C. User supplied termination)

III. PRIMITIVE COMMANDS

1.0 System- and File-Oriented Commands

LO - LOgon, LOgoff

2.0 Record-Oriented Commands

A,AD <name> - Add - add a record of the named type to the file. The record is added immediately after the current record, or in place of the record just deleted (if any), or at the end of the file if no record has yet been accessed.

I,IN <name> - Insert - insert a record of the named type into the file. The record is inserted immediately before the current record, or in place of the record just deleted (if any), or at the beginning of the file if no record has yet been accessed.

EX - EXclude - exclude the current record from the working subset.

D,DE - Delete - delete the current record from the file.

R,RE,RL - Release - expand the working subset to include all allocated records.

S,SE (<bexpr>)

- Select - selects a working subset of the file based on (Boolean combinations of) relational or logical values of field(s).

<bexpr> ::= <bvalue> | (<bexpr>) | <uop><bexpr> | <bexpr><bop><bexpr>

<bvalue> ::= (<name><rop><value>) | <name>

<uop> ::= NOT | ¬

<bop> ::= & | AND | OR | |

<rop> ::= = | EQ | > | GT | LT | < | NE | LE | GE

<value> ::= <string> | '<string>'

CR - Current Record - access the current record in the working subset. (An effective NOP.)

NR - Next Record - access the next record in the working subset.

F,FO <number>

- Forward - access the nth subsequent record in the working subset.

PR - Previous Record - access the previous record in the working subset.

B,BA <number> - Back - access the nth previous record in the working subset.

FR - First Record - access the first record in the working subset.

LR - Last Record - access the last record in the working subset.

3.0 Page-Oriented Commands

CP - Current Page - (re)display the current page.

NP - Next Page - display the next page in the current record.

PP - Previous Page - display the previous page in the current record.

FP - First Page - display the first page in the current record.

LP - Last Page - display the last page in the current record.

P,PA <name> - Page - display the page in the current record with the given name.

P,PA <number> - Page - display the nth page in the current record.

4.0 Field-Oriented Commands

CF - Current Field - access the current field in the current page.

NF - Next Field - access the next field in the current page.

PF - Previous Field - access the previous field in the current page.

FF - First Field - access the first field in the current page.

LF - Last Field - access the last field in the current page.

<name> - access the named field in the current page.

<number> - access the nth field in the current page.

<touch> - access the field touched (if touch sensitive).

5.0 Special Key Functions

<CTRL-C> - terminate execution.

<CTRL-U> - throw away the current input line.

<CTRL-Z> - when modifying a field, leave the field unchanged; otherwise, leave the current record unchanged.

<rubout> - delete the last character in the current line.

IV. FILE STRUCTURE

1.0 Introduction

A logical file in the generic system consists of a physical data file and a physical status file. The two files are managed as a single entity by the generic system.

2.0 Data File

The data file is a random access file in which the data associated with a logical file is maintained by the generic system. All records in the file are the same size, each record being a multiple of 256 words. The file consists of a control record followed by the data records. The data records consist of two subsets of contiguous records (either, but not both, of which may be empty); the first subset are records which are in use (allocated or deleted); the second subset are records which have never been used.

2.1 Control Record

The first physical record of a data file is reserved for the maintenance of the file by the generic system. The format of the

record is as follows:

Word 1 holds the number of records in the file.

Word 2 holds the physical record number of the first unused record in the file.

Word 3 holds the record size plus one of the records in the file.

Word 4 holds the physical record number of the first record on the allocated list.

Word 5 holds the physical record number of the first record on the deleted list.

Word 6 holds the physical record number of the last record on the allocated list.

Word 7 holds the number of allocated records in the file.

The remaining words are unused and have undefined values.

2.2 Data Records

The first two words of each data record are reserved for future changes to the system. The third word of each data record

contains the (coded) record type of the record. The remaining words contain the actual data associated with the record. Any excess words in the record, which must be a multiple of 256 words in length, are unused and have undefined values. All contents of a data record are undefined if the record is deleted or unused.

3.0 Status File

The status file is organized as a random-access file of 256 word records. Each record's format is identical to that of the STATUS area, (described in section V.6), with the exception that the contents of the final word are unpredictable. The allocated data records are externally chained on a doubly-linked list maintained in the STATEL status elements in the status file. The deleted data records are externally chained on a singly-linked list of STATEL status elements. The included/excluded status of each allocated record in up to 15 working sets is recorded in the status file with both deleted and unused records marked as if they were deleted. Excess STATEL status elements in the final record of the status file are initialized as if for deleted records. There is no STATEL status element for the control record of the data file, so the physical position in the status file of a STATEL status element associates it with the corresponding logical - rather than physical - record in the data file.

V. DATA ORGANIZATION

1.0 Introduction

The structure of this description of the data organization is the same as the structure of the data organization itself. Thus by reading through linearly, one can see the organization as a whole; or one can use the index to go directly to any block in question. One point to note is that all definitions of files, records, and pages are kept in memory, and that only the data in a given record is swapped in and out. The system is entirely table-driven, and unless otherwise noted all variables are two-bytes in size (integer), and all indices are in units of two-bytes.

2.0 MEMORY

MEMORY is at the same time the name of the universal array in which all system tables and data are kept, and the name of the block of nine words that appear at the beginning of that array. The uses of those particular nine words are as follows:

MEMORY(1) holds the index in MEMORY of the BUFCTL area. It must be initialized to that value.

MEMORY(2) holds the index in MEMORY of the CSTATE area. Again, this value must be initialized.

MEMORY(3) holds the length of MEMORY (as the universal array). It is needed only by a debugging routine that dumps MEMORY, and if that routine (DEFDMP) is to be used, MEMORY(3) must be initialized to that length. Normally, however, this location may be ignored.

MEMORY(4) holds the font width. It must be given that value initially.

MEMORY(5) is currently unused, but is reserved for access to multiple files.

MEMORY(6) is currently unused, but is reserved for access to multiple files.

MEMORY(7) is currently unused, but is reserved for access to multiple files.

MEMORY(8) is currently unused, but is reserved for allocation of logical I/O units in support of multiple files.

MEMORY(9) is currently unused, but is reserved for management of multiple users.

3.0 User Data

3.1 CSTATE

The system maintains, (and under a multi-user version would maintain for each user), a block called CSTATE, which describes the current user state. The multiple user access arrangement is not yet defined. The use of each of the ten words in CSTATE are as follows:

CSTATE(1) holds the physical record number of the first record on the included list. It is zero if there are no included records.

CSTATE(2) holds the physical record number of the last record on the included list. It is zero if there are no included records.

CSTATE(3) holds the index in CSTATE of the user's ACCBLK, and should be initialized to that value.

CSTATE(4) holds the index in MEMORY of the user's RECORD area, and should be initialized to that value.

CSTATE(5) holds the index in MEMORY of FLDSCR. Under a multi-file version, it would be initialized to zero, and given the index value when a file was first accessed. Currently, it should be given the index value initially.

CSTATE(6) holds the index in FLDSCR of RDESCR. It is given the index value when a record is accessed, and is zero when no record is currently accessed. Under a multi-file version, it would be zeroed by the system each time a new file was accessed.

CSTATE(7) holds the index in RDESCR of PDESCR. It is zeroed each time a record is accessed, and given the index value each time a page is accessed.

CSTATE(8) holds the current field number. It is zeroed each time a page is accessed, and assigned the field number each time a field is accessed.

CSTATE(9) holds the index in MEMORY of the next CSTATE area or zero if this is the last CSTATE area.

CSTATE(10) holds the working set number currently in use. It must be initialized to zero.

3.2 ACCBLK

The generic data transaction system maintains within this block accumulated and current measurements of the interaction of the operator with the system. The specific words are used as follows:

ACCBLK(1) holds the stroke count for the most recently typed

entry.

ACCBLK(2) holds the rubout count for the most recently typed entry.

ACCBLK(3) holds the line cancel count for the most recently typed entry.

ACCBLK(4) holds the response time of the user to the most recent prompt (in 60 hertz ticks).

ACCBLK(5) holds the typing time of the most recent entry (in 60 hertz ticks).

ACCBLK(6) holds the elapsed time of the processing of the most recent directive (in 60 hertz ticks).

ACCBLK(7) holds the number of commands processed by the system.

ACCBLK(8) holds the number of errors processed by the system.

4.0 File Definition(s)

In the current implementation, there is but one (logical) file, which must be accessed by the user (with the procedure UINITL). However the data organization allows any number of files to be defined, each definition being a block as described in this section, multiple definitions being arranged sequentially in MEMORY. The multiple file access arrangement is not yet defined.

4.1 FLDSCR

The file descriptor, FLDSCR, is a block of twelve words which are used as follows:

FLDSCR(1) holds the index in FLDSCR of RNDSCR. It must be initialized to that value.

FLDSCR(2) holds the index in FLDSCR of RTTABL. Again, this value must be initialized.

FLDSCR(3) holds the maximum number of records in the file, including the control record, deleted records, and unused records. The value is extracted from the file control record when the file is opened.

FLDSCR(4) holds the physical record number of the first unused

record in the file. The value is extracted from the file control record when the file is opened. If there are no unused records, this location equals FLDSCR(3)+1.

FLDSCR(5) holds the record size plus one of records in the file, and must be appropriately initialized.

FLDSCR(6) holds the physical record number of the first record on the allocated list. It is zero if there are no allocated records. The value is extracted from the file control record when the file is opened.

FLDSCR(7) holds the physical record number of the first record on the deleted list. It is zero if there are no deleted records. The value is extracted from the file control record when the file is opened.

FLDSCR(8) holds the physical record number of the last record on the allocated list. It is zero if there are no allocated records. The value is extracted from the file control record when the file is opened.

FLDSCR(9) is the number of allocated records in the file. The value is extracted from the file control record when the file is opened.

FLDSCR(10) holds the physical record number of the file status

record currently in the STATUS area.

FLDSCR(11) holds the index in MEMORY of the STATUS area for the file. It should be initialized to zero.

FLDSCR(12) is currently unused, but is reserved for accessing multiple working sets.

4.2 RNDSCR

RNDSCR is a block of two words describing the record name symbol table, a combination of RNTABL and RNIMAG.

RNDSCR(1) holds the index in RNDSCR of RNIMAG, or zero if RNDSCR(2) is zero.

RNDSCR(2) holds the number of record names in the table.

4.3 RNTABL

RNTABL and RNIMAG together constitute the record name symbol table. RNTABL describes the name images in RNIMAG, and holds the record type for each record name. A record type may be unnamed, have one or many names.

RNTABL(1,n) holds the (coded) record type. Record types are (contiguous) positive integers used as indices into RTTABL.

RNTABL(2,n) holds the index in RNIMAG of the (first two characters of the) nth record name.

RNTABL(3,n) holds the length in characters of the nth record name.

4.4 RNIMAG

RNIMAG is a packed string of record type names. Each name is begun on a word boundary.

4.5 RTTABL

RTTABL, the record type table, links a record type with its RDESCR.

RTTABL(n) holds the index in FLDSCR of the RDESCR for record type number n.

4.6 Record Definition(s)

Record definitions are repeatable within a given file

definition, just as file definitions are repeatable within MEMORY. Multiple record definitions are concatenated at this point in the file definition, and are located through RTTABL.

4.6.1 RDESCR

The record descriptor, RDESCR, is a block of five words which are used as follows:

RDESCR(1) holds the index in RDESCR of the first PDESCR.

RDESCR(2) holds the index in RDESCR of SMDSCR.

RDESCR(3) holds the index in RDESCR of FORMAT.

RDESCR(4) holds the index in RDESCR of DDESCR.

RDESCR(5) holds the index in RDESCR of PGDSCR.

4.6.2 SMDSCR

SMDSCR is a block of two words describing the field name symbol table, a combination of SMTABL and SIMAGE.

SMDSCR(1) holds the index in SMDSCR of SIMAGE, or zero if

SMDSCR(2) is zero.

SMDSCR(2) holds the number of symbols in the table.

4.6.3 SMTABL

SMTABL and SIMAGE together constitute the field name symbol table. SMTABL describes the name images in SIMAGE, and points to the entry in DTABLE for each field named. Fields may have no name, one, or many names.

SMTABL(1,n) holds the index (second subscript) of the entry for the field in DTABLE.

SMTABL(2,n) holds the index in SIMAGE of the (first two characters of the) nth field name.

SMTABL(3,n) holds the length in characters of the nth field name.

4.6.4 SIMAGE

SIMAGE is a packed string of field names. Each name is begun on a word boundary.

4.6.5 DDESCR

DDESCR is one word describing DTABLE.

DDESCR(1) holds the number of fields described in DTABLE, i.e., the length of DTABLE.

4.6.6 DTABLE

There is one entry in DTABLE for each unique field in a record. (A given field, social security number for example, might appear on several pages of the same record.) It points to the DATA entry, and describes how the data is displayed, for each field.

DTABLE(1,n) holds the index in DATA of the current contents of the field.

DTABLE(2,n) holds the word length of the field contents in DATA.

DTABLE(3,n) holds the index in FORMAT of the display format of the field.

DTABLE(4,n) holds the character (byte) length of the echo area for the field. If the field length is longer than the echo length, the extra characters are written over the last position in the echo field. This length must include room for the blank which is

prefixed to all printed fields.

DTABLE(5,n) is currently unused, but is reserved for password access to the field.

DTABLE(6,n) contains a non-zero flag which is passed to an application specific UCHECK (user check) routine, to verify the correctness of a new value being given to the field. A zero flag indicates that no verification is required.

4.6.7 FORMAT

FORMAT is a packed string of format images for the data for each field in DTABLE. Each object time format string is begun on a word boundary. The display of blank fields is suppressed unless a blank precedes the left parenthesis in the format string.

4.6.8 PGDSCR

PGDSCR is a block of two words describing the page name symbol table, a combination of PGTABL and PGIMAG.

PGDSCR(1) holds the index in PGDSCR of PGIMAG, or zero if PGDSCR(2) is zero.

PGDSCR(2) holds the number of symbols in the table.

4.6.9 PGTABL

PGTABL and PGIMAG together constitute the page name symbol table. PGTABL describes the name images in PGIMAG, and holds the page number for each name. Pages may have no name, one, or many names.

PGTABL(1,n) holds the page number for the nth page name.

PGTABL(2,n) holds the index in PGIMAG of the (first two characters of the) nth page name.

PGTABL(3,n) holds the length in characters of the nth field name.

4.6.10 PGIMAG

PGIMAG is a packed string of page names. Each name is begun on a word boundary.

4.6.11 Page Definition(s)

Page definitions are repeatable within a given record

definition - just as record definitions are repeatable within file definitions - multiple page definitions being concatenated at this point in the record definition. The first page definition is located through RDESCR, and subsequent ones are found by traversing the links in the PDESCR for each page.

4.6.11.1 PDESCR

The page descriptor, PDESCR, is a block of six words which are used as follows:

PDESCR(1) holds the page number.

PDESCR(2) holds the index in RDESCR of the PDESCR for the next page. A value of zero indicates no next page.

PDESCR(3) holds the index in RDESCR of the PDESCR for the previous page. A value of zero indicates no previous page.

PDESCR(4) holds the index in PDESCR of SDESCR.

PDESCR(5) holds the index in PDESCR of TDESCR.

PDESCR(6) holds the index in PDESCR of FDESCR.

4.6.11.2 TDESCR

TDESCR is a single word describing TTABLE.

TDESCR(1) holds the number of touch areas defined in TTABLE, i.e., the length of TTABLE.

4.6.11.3 TTABLE

Each entry in TTABLE defines one touch area for the page. There may be no, or any number of, touch areas defined.

TTABLE(1,n) holds the index (second subscript) in FTABLE of the field referenced by a touch in area n.

TTABLE(2,n) holds the x screen coordinate of the upper left corner of touch area n.

TTABLE(3,n) holds the y screen coordinate of the upper left corner of touch area n.

TTABLE(4,n) holds the x screen coordinate of the lower right corner of touch area n.

TTABLE(5,n) holds the y screen coordinate of the lower right corner of touch area n.

4.6.11.4 FDESCR

FDESCR is a single word describing FTABLE.

FDESCR(1) holds the number of fields on the page, which is also the length of FTABLE. A page may contain no fields.

4.6.11.5 FTABLE

There is one entry in FTABLE for each field on the page. It gives the position on the page, and a pointer to the description of the data, for each field. Note that a blank is prefixed to all printed fields.

FTABLE(1,n) holds the index (second subscript) in DTABLE for field number n.

FTABLE(2,n) holds the x screen coordinate of the lower left corner of the first print position in the field.

FTABLE(3,n) holds the y screen coordinate of the lower left corner of the first print position in the field.

4.6.11.6 SDESCR

SDESCR is a block of two words describing SSTABL and SSTRNG, which together define the static elements of the page.

SDESCR(1) holds the index in SDESCR of SSTRNG, or zero if SDESCR(2) is zero.

SDESCR(2) holds the number of entries (commands) in SSTABL. This table may be empty.

4.6.11.7 SSTABL

Each entry in SSTABL is a command to either 1) move the pointer to a given position on the screen, or 2) draw a line from the current pointer position to a given position, (leaving the pointer at the end position), or 3) to write at a given position one of the character strings in SSTRNG.

SSTABL(1,n) holds the x screen coordinate of the target point.

SSTABL(2,n) holds the y screen coordinate of the target point.

SSTABL(3,n) may have any of the following values: 1) zero, which commands that the pointer be moved to the target point; 2) two, which commands that a line be drawn from the current pointer

position to the target point; or 3) minus the index in SSTRNG of (the first two characters of) a string to be written on the screen, with the target point being the position of the lower left corner of the first character.

4.6.11.8 SSTRNG

SSTRNG is a packed string of static character strings for the page. Each string is begun on a word boundary, and must be terminated by a zero byte.

5.0 Data Record I/O Areas

Storage must be reserved at this point for the data record I/O areas. A single data record I/O area must be provided for each user. Since all indexing is relative to the data record I/O area referenced by CSTATE(4) for the current user, only the format of one (the first) relocatable data record I/O area is described.

5.1 RECORD

RECORD is a block of five words containing information, used and maintained by the system, about the record whose data is currently in the DATA area immediately following RECORD. The last three words of RECORD are the first three of the physical block of data for the record.

RECORD(1) holds a flag indicating whether or not changes have been made to the record since it was last read into memory (or saved). A value of one indicates change, zero indicates none.

RECORD(2) holds the sequential, physical record number of the record in its physical, direct file. Zero indicates that no record has been accessed from the current file.

RECORD(3) holds the physical record number of the next logical record on the allocated list.

RECORD(4) holds the physical record number of the previous logical record on the allocated list.

RECORD(5) holds the (coded) record type of the record. A positive value is the index (second subscript) of the entry for this record's type in RTTABL. A value of zero indicates that the record has been permanently, logically deleted from the file.

5.2 DATA

DATA is a concatenated string of the data currently in the record. Each data item is begun on a word boundary. All templates for describing the data arrangements of the various record data types for all of the files must be specified using this same addressing origin.

6.0 Status Record I/O Areas

Storage may be reserved at this point for any number of status record I/O areas, which are allocated for I/O using a least-recently-used (LRU) algorithm. It is suggested that at least four status record I/O areas be provided for each file or each user (whichever are fewer).

6.1 STATUS

STATUS is a status record I/O area. It contains a single, three-word status element (STATEL) for each data record in the logical file. STATUS is 256 words long, and contains 85 status elements. The final word of STATUS is set to indicate that the status record has been modified and must be written.

6.2 STATEL

Each STATEL status element is used to maintain the status and links of a single data record. All STATEL are initialized by the file creation/maintenance utility, and are of the following format:

STATEL(1) holds the status of the nth+1 physical record in the data file, where n is the status element's sequential position in

the status file. A value of -1 implies that the data record is either deleted or unused; any other value implies the record is allocated. When STATEL(1) indicates the data record is allocated, the data record's status in the working set indexed by CSTATE(10) is determined by testing the bit zero-indexed from the least significant bit by CSTATE(10). If the bit's value is zero, the data record is included in the corresponding working set; a value of 1 means the record is excluded. (For example, if STATEL(1)=5, the corresponding data record is excluded from the first and third working sets only.)

STATEL(2) holds the physical record number of the data record next on the same list (i.e., either allocated or deleted). If the data record is unused, i.e. is on neither list, the value of this location is meaningless.

STATEL(3) holds the physical record number of the data record previous on the allocated list. If the record is not allocated, the value of this location is meaningless.

7.0 BUFFER CONTROL

BUFCTL and BUFTBL together make up the control structure for the least-recently-used (LRU) management of the multiple buffers in the STATUS area.

7.1 BUFCTL

The three words in BUFCTL are defined and must be initialized as follows:

BUFCTL(1) holds the index in BUFCTL of the control element for the newest buffer in STATUS.

BUFCTL(2) holds the index in BUFCTL of the control element for the oldest buffer in STATUS.

BUFCTL(3) holds the index in BUFCTL of the control element of the most recently allocated dedicated buffer in STATUS. It must be initialized to zero.

7.2 BUFTBL

BUFTBL contains one control element for each of the multiple buffers in the STATUS area. A BUFTBL element is defined and must

be initialized as follows:

BUFTBL(1,n) holds the index in BUFCTL of the control element for the next older buffer in STATUS.

BUFTBL(2,n) holds the index in BUFCTL of the control element for the next newer buffer in STATUS.

BUFTBL(3,n) holds the index in MEMORY of the buffer in STATUS controlled by the element.

BUFTBL(4,n) holds the number of words in a status file record to be rewritten and must be initialized to zero.

BUFTBL(5,n) holds the disk unit on which the status file block will reside and must be initialized to zero.

BUFTBL(6,n) holds the file block number of the status file block and must be initialized to zero.

VI. COMMON BLOCKS

Most communication between procedures is accomplished through argument lists, but in a few special cases common blocks are used. For a description of how to tailor these common blocks for a specific application, please see the User's Guide. All common blocks except ZTOUCH will be merged into other parts of ZMEMRY with the expansion to multiple users.

ZFLAGS

FIELD - if one, indicates that a field has been addressed and a field update needs to be done; is zero otherwise.

RECCNT - the count of records in the working set (included list).

OLDTIM - the low order value of the system clock when the last command was entered.

LOGTOG - if one, indicates that the user has executed a LOGON command; -1 initially and after the execution of a LOGOFF command.

Used by Procedures: ADD, BLOCK DATA, CMNDEX, CMNDX1, CMNDX2, ENDTIM, FLDUPD, (main), RELEAS, REMOVE, SAMPLE, TPANEL, WRAPUP

ZLOC

CMND - y coordinate of command echo position; (x coordinate = 0); may have any value 16n, 0 <= n <= 31.

DIAG - y coordinate of diagnostics echo position; (x coordinate = 0); may have any value 16n, 0 <= n <= 31.

X1,Y1,X2,Y2 - coordinates of diagonally opposite corners of the input echo window; each may have any value n, 0 <= n <= 511.

Used by Procedures: BLOCK DATA, ERROR, (main), MESSAG, SDISPL

ZMEMRY

MEMORY - see section III.1.1.

Used by Procedures: FLDUPD, (main), SCAN, TPANEL, WRAPUP

ZTOUCH

This common block is used to pass touch coordinates from the device handler to TPANEL, which handles the touch pseudo-commands.

X,Y - coordiantes of touch; each may have any value n, 0 <= n <= 31.

Used by Procedures: TPANEL

ZUNIT

UNIT - the disk unit on which record-data blocks reside.

PUNIT - the plasma panel.

IUNIT - the keyboard.

SUNIT - the disk unit on which the file status blocks reside.

Used by Procedures: BLOCK DATA, DATAIO, ERROR, FDISPL, INITL, (main), MESSAG, RITE, SELERR, SELER1, SELER2, STATIO, WRAPUP

VII. PROCEDURE DESCRIPTIONS

ADD

This function adds a new record to the current file, inserting it optionally before or after the current record in the allocated list, and accesses it so that data may be entered.

Common Blocks Used: ZFLAGS

Called by: CMNDX1

Calls: DATAIO, GETPTR, GETREC, INSTAL, LOOKUP, STATUS

AKGET (assembly)

This subroutine is used to extract the operator performance measurements from the device handler.

Called by: SAMPLE

Calls: -

AKSET (assembly)

This subroutine is used to specify a synchronous extension to the plasma panel character generator for time delay purposes.

Called by: INITL

Calls: -

ARITH

This function performs an arithmetic comparison of input (character) data with stored data.

Called by: COMPAR

Calls: -

BLOCK DATA

BLOCK DATA initializes the common blocks ZFLAGS, ZLOC, and ZUNIT.

Common Blocks Used: ZFLAGS, ZLOC, ZUNIT

BOOLN (assembly)

This function evaluates boolean expressions (from tables generated by BSCAN and RECTST).

Called by: SELECT

Calls: -

BSCAN

This function performs syntax analysis of (complex) relational expressions.

Called by: SELECT

Calls: SYMBOL, TOKEN

BUFGET

This subroutine removes a buffer control element from the doubly-linked LRU queue.

Called by: STATIO

Calls: -

BUFPUT

This subroutine installs the most recently used buffer control element onto the doubly-linked LRU queue.

Called by: STATIO

Calls: -

BUFWRT

This subroutine physically writes a buffer if it is appropriately flagged in its buffer control element.

Called by: STATIO

Calls: -

BULK (assembly)

This subroutine erases (or writes) solid, rectangular blocks on the plasma screen.

Called by: ERASE, INITL, LOGOFF, LOGON, SDISPL

Calls: -

CMNDEX

This subroutine drives execution of the command primitives.

Common Blocks Used: ZFLAGS

Called by: SCAN

Calls: CMNDX1, CMNDX2, ERROR

CMNDIN

This function scans an input string to determine what command primitive (if any) the string contains.

Called by: SCAN

Calls: TEXT

CMNDX1

This subroutine drives the excution of all record-oriented command primitives.

Common Blocks Used: ZFLAGS

Called by: CMNDEX

Calls: ADD, DATAIO, ERROR, GETPAG, GETREC, LOGOFF, LOGON, MESSAG, PDISPL, RELEAS, REMOVE, SELECT, VALUE

CMNDX2

This subroutine drives the execution of all page- and field-oriented command primitives.

Common Blocks Used: ZFLAGS

Called by: CMNDEX

Calls: ERROR, FLDNAM, GETFLD, GETPAG, PAGNAM, PDISPL, VALUE

COMPAR

This function evaluates the truth or falsity of relational expressions passed to it.

Called by: RECTST

Calls: ARITH, MATCH

CONOFF

This subroutine disables all keyboard input from the system console.

Called by: INITL

Calls: -

CONON

This subroutine re-enables the keyboard input from the system console.

Called by: WRAPUP

Calls: -

CVT

This function converts an input character string to internal integer format.

Called by: VALUE

Calls: -

CVTFLD

This function converts and copies an entered string into its assigned record data field.

Called by: FLDUPD

Calls: -

DATAIO

This subroutine performs data record read/write operations.

Common Blocks Used: ZUNIT

Called by: ADD, CMNDX1, GETREC, LOGOFF, LOGON

Calls: -

DISPLA (assembly)

This subroutine draws lines on the plasma screen or simply moves the graphics cursor.

Called by: SDISPL

Calls: -

ELINE (assembly)

This subroutine moves the write-position on the plasma screen to the beginning of a given horizontal line (on a multiple of sixteen boundary).

Called by: (main), ERROR, MESSAG

Calls: -

ENDTIM

This subroutine updates the current user's user-performance data area with the elapsed execution time (in 60Hz clock ticks) of the most recent directive.

Common Blocks Used: ZFLAGS

Called by: (main)

Calls: TDIF, TIMGET

ERASE

This subroutine erases a line of text on the display screen.

Called by: (main), ERROR, MESSAG

Calls: BULK

ERROR

This subroutine writes (or causes to be written) all error messages.

Common Blocks Used: ZLOC, ZUNIT

Called by: CMNDEX, CMNDX1, CMNDX2, FLDUPD, SCAN

Calls: ELINE, ERASE, SELERR, UERROR

FDISPL

This subroutine writes one or all dynamic elements of a page.

Common Blocks Used: ZUNIT

Called by: PDISPL

Calls: MOVCUR, RITE, TIME

FLDNAM

This function accesses the field of the current page with a given name.

Called by: CMNDX2

Calls: GETFLD, SLOOK

FLDUPD

This subroutine checks the validity of a new field value, and drives the rest of the field update process.

Common Blocks Used: ZFLAGS, ZMEMRY

Called by: (main)

Calls: CVTFLD, ERROR, PDISPL, UCHECK

GETBIT

This function returns the included/excluded/deleted status of a record.

Called by: GETREC, INSTAL, RELEAS

Calls: IAND, ISHFTL, ISHFTR, STATUS

GETFLD

This function moves the cursor to the next field, previous field, last field, or field number n, of the current page.

Called by: CMNDX2, FLDNAM

Calls: MOVCUR

GETPAG

This function displays the next page, previous page, last page, or page number n, of the current record.

Called by: CMNDX1, CMNDX2, PAGNAM

Calls: -

GETPTR

This function returns the physical record number of the record next or previous on the allocated list.

Called by: ADD, GETREC, INSTAL

Calls: STATUS

GETREC

This function accesses the first record, next record, previous record, nth subsequent record, nth previous record, or the last record in the current file.

Called by: ADD, CMNDX1, REMOVE, SELECT

Calls: DATAIO, GETBIT, GETPTR, MESSAG

IAND (assembly)

This function returns the bit by bit "and" of all its arguments.

Called by: GETBIT, PUTBIT

Calls: -

INITL

This subroutine establishes file definitions.

Common Blocks Used: ZUNIT

Called by: (main)

Calls: AKSET, BULK, CONOFF, UINITL, OTSWA

INSTAL

This subroutine links a new record into the allocated list.

Called by: ADD

Calls: GETBIT, GETPTR, PUTPTR

IOR (assembly)

This function returns the bit by bit "or" of all its arguments.

Called by: PUTBIT

Calls: -

ISHFTL (assembly)

This function returns the specified left shift of a value.

Called by: GETBIT, PUTBIT

Calls: -

ISHFTR (assembly)

This function returns the specified right shift of a value.

Called by: GETBIT

Calls: -

LOGOFF

This subroutine erases the plasma screen, deactivates the touch panel, updates the file control record, and saves the current record if it has been changed. (It also prevents further use of the system by the current user until a LO command is executed.)

Called by: CMNDX1, WRAPUP

Calls: BULK, DATAIO, STATUS, TOUCH, ULOG

LOGON

This subroutine initializes the included list, erases the plasma screen, and activates the touch panel. (It also enables use of the system by the current user.) The system is left in the no-record-yet-accessed condition.

Called by: CMNDX1

Calls: BULK, DATAIO, MESSAG, OFFTP, RELEAS, TOUCH, ULOG

LOOKUP

This function performs a standard symbol-table look-up.

Called by: ADD, PAGNAM, RECTST, SLOOK

Calls: MATCH

(main)

This procedure first calls INITL to initialize the files; then loops to pick-up entered strings and call CMNDEX or FLDUPD as appropriate.

Common Blocks Used: ZFLAGS, ZLOC, ZMEMRY, ZUNIT

Called by: (monitor)

Calls: ELINE, ENDTIM, ERASE, FLDUPD, INITL, OFFTP, ONTP, PNWAIT, SAMPLE, SCAN, UPRMPT

MATCH

This function compares two character strings of equal length for equality.

Called by: COMPAR, LOOKUP, SYMBOL, TOKEN

Calls: -

MESSAG

This subroutine writes/erases informative messages.

Common Blocks Used: ZLOC, ZUNIT

Called by: CMNDX1, GETREC, LOGON, SELECT

Calls: ERASE, ELINE, UMSG

MOVCUR (assembly)

This subroutine moves the I/O cursor to a specified screen location.

Called by: FDISPL, GETFLD, TPANEL

Calls: -

NOT (assembly)

This function returns the bit by bit complement of a value.

Called by: PUTBIT

Calls: -

OFFTP (assembly)

This subroutine inhibits touch panel interrupts.

Called by: LOGON, (main)

Calls: -

ONTP (assembly)

This subroutine arms the touch panel for further interrupts.

Called by: (main)

Calls: -

OTSWA (assembly)

This subroutine is used to control object time system work area allocations.

Called by: INITL

Calls: -

PAGNAM

This function causes the display of the page (of the current record) having a given name.

Called by: CMNDX2

Calls: GETPAG, LOOKUP

PDISPL

This subroutine causes to be written on the plasma screen either an entire page, or the dynamic part of eithe an entire page or a single field.

Called by: CMNDX1, CMNDX2, FLDUPD

Calls: FDISPL, SDISPL

PNKILL (assembly)

This subroutine disables any active output operations to the plasma panel.

Called by: WRAPUP

Calls: -

PNWAIT (assembly)

This subroutine waits for all queues on the plasma panel to quiesce.

Called by: (main), TIME, TPANEL, WRAPUP

Calls: -

PUTBIT

This subroutine updates the included/excluded/deleted status of a record.

Called by: RELEAS, REMOVE

Calls: IAND, IOR, ISHFTL, NOT, STATUS

PUTPTR

This subroutine updates the pointer to the record next or previous on the allocated list.

Called by: INSTAL, REMOVE

Calls: STATUS

PWRITE (assembly)

This subroutine writes a given character string at a given location on the plasma screen.

Called by: SDISPL

Calls: -

RECTST

This subroutine examines the current record to evaluate the truth or falsity of relational expressions passed to this subroutine in tabular form.

Called by: SELECT

Calls: COMPAR, LOOKUP, UTEST

RELEAS

This function puts all allocated records (in the current file) in all working sets.

Common Blocks Used: ZFLAGS

Called by: CMNDX1, LOGON

Calls: GETBIT, PUTBIT

REMOVE

This function removes the current record from the current working set, and depending on the value of an input flag, may delete the record from the file as well. The system is left in the no-record-accessed condition.

Common Blocks Used: ZFLAGS

Called by: CMNDX1, SELECT

Calls: GETREC, PUTBIT, PUTPTR

RITE

This subroutine is called only by FDISPL, and exists solely so that the write of each field on the plasma screen may be done without subscripting a format variable, (which is syntactically illegal).

Common Blocks Used: ZUNIT

Called by: FDISPL

Calls: -

SAMPLE

This subroutine is used to sample the user performance measurements collected by the device handler.

Common Blocks Used: ZFLAGS

Called by: (main)

Calls: AKGET, TDIF

SCAN

This subroutine drives the interpretation and execution of commands (except for those pseudo-commands handled by FLDUPD and TPANEL).

Common Blocks Used: ZMEMRY

Called by: (main)

Calls: CMNDEX, CMNDIN, ERROR, USCAN

SCREEN (assembly)

This function converts a touch panel coordinate into a corresponding screen coordinate.

Called by: TLOOK

Calls: -

SDISPL

This subroutine drives the display of the static elements of a page on the plasma screen.

Common Bloccks Used: ZLOC

Called by: PDISPL

Calls: BULK, DISPLA, PWRITE, TIME

SELECT

This function drives execution of the select command, which selects a subset of the current working set to be the new working set, based on (complex) relational expressions entered by the user. The system is left in the no-record-yet-accessed condition.

Called by: CMNDX1

Calls: BOOLN, BSCAN, GETREC, MESSAG, RECTST, REMOVE

SELERR

This subroutine initiates the writing of error messages for errors detected during syntax analysis of the select command.

Common Blocks Used: ZUNIT

Called by: ERROR

Calls: SELER1, SELER2

SELER1

This subroutine writes part of the set of error messages for errors detected during syntax analysis of the select command.

Common Blocks Used: ZUNIT

Called by: SELERR

Calls: -

SELER2

This subroutine writes part of the set of error messages for errors detected during syntax analysis of the select command.

Common Blocks Used: ZUNIT

Called by: SELERR

Calls: -

SLOOK

This function translates an input field name into a pointer to the descriptor of the field referenced.

Called by: FLDNAM

Calls: LOOKUP

STATIO

This subroutine performs LRU buffered I/O of file status records.

Common Blocks Used: ZUNIT

Called by: STATUS

Calls: BUFGET, BUFPUT, BUFWRT

STATUS

This subroutine swaps the status records to allow processing of the status information for any given data record.

Called by: ADD, GETBIT, GETPTR, LOGOFF, PUTBIT, PUTPTR

Calls: STATIO

SYMBOL

This function assigns relational expressions to the table which is used during the evaluation phase of the select command.

Called by: BSCAN

Calls: MATCH

TDIF (assembly)

This function computes the unsigned difference (modulo 65536) between two low order system clock 60 hertz tick counts.

Called by: ENDTIM, SAMPLE

Calls: -

TEXT

This function extracts the next symbol from a command line, and returns the first two characters of that symbol.

Called by: CMNDIN

Calls: -

TIME

This subroutine synchronizes use of the user's UTIME subroutine to prevent non-reentrancy problems.

Called by: FDISPL, SDISPL

Calls: PNWAIT, UTIME

TIMGET (assembly)

This function returns the current low order word of the line frequency clock.

Called by: ENDTIM

Calls: -

TLOOK

This function translates touch coordinates into a pointer to the descriptor of the field touched.

Called by: TPANEL

Calls: SCREEN

TOKEN

This function extracts and classifies the next token in the relational string part of a select command line.

Called by: BSCAN

Calls: MATCH

TOUCH (assembly)

This subroutine activates/deactivates the touch panel.

Called by: LOGOFF, LOGON, WRAPUP

Calls: -

TPANEL

This subroutine is the touch panel interrupt completion routine.

Common Blocks Used: ZFLAGS, ZMEMRY, ZTOUCH

Called by: (monitor)

Calls: MOVCUR, PNWAIT, TLOOK, UTIME, UTOUCH

VALUE

This function extracts the next symbol from a command line, and tries to convert it to an integer format.

Called by: CMNDX1, CMNDX2

Calls: CVT

WRAPUP

This subroutine is invoked during program termination to allow device deactivation.

Common Blocks Used: ZFLAGS, ZMEMRY, ZUNIT

Called by: (main)

Calls: CONON, LOGOFF, PNKILL, PNWAIT, TOUCH, UWRAP

VIII. FILE CREATION/MAINTENANCE UTILITY

The file creation/maintenance utility for the Generic Data Transaction System consists of two parts. The first, FILUPD, compacts the user's disk and then starts the second part. The second part, NPRDCU, is a separate program that lets the user perform any of four activities. Each of these activities operates on one logical file at a time. Each logical file consists of a physical data file and a physical status file as described in Part IV of this document.

The four activities available are: 1) to create a logical file; 2) to increase the capacity of a logical file; 3) to check a logical file for internal consistency; and 4) to reorder the data records (in a logical file) so that their logical and physical orders are the same. If the user chooses to reorder a file (activity four), a check for consistency (activity three) is automatically done first.

The check for consistency is provided because an abnormal termination of execution of the generic system could leave changes to the logical file only partially recorded in the permanent disk files. In the event of such an abnormal termination, a consistency check can be performed to determine whether or not the logical file(s) in use at the time need to be restored from back-up copies. A consistency check is automatically done before a file is reordered because internal consistency is necessary for the reordering operation to function correctly.

The I/O operations performed by the generic system are executed most efficiently when the logical and physical orders of the data records are identical. If many data records have been added to and deleted from a given file since it was created, the records' logical order may differ significantly from their physical order. In such a case, a reordering of the data records (activity four) could significantly improve the performance of the generic system. It is for this reason that the reordering option was included.

IX. APPLICATION SPECIFIC PROCEDURES

Application specific procedures may exist as user supplied software appendages at strategic points in the generic data transaction system. It is anticipated that the procedures will be used to customize the operation of the generic system in such a way as to either modify or extend the capabilities required by the user's specific application. The procedures must exist as FORTRAN IV callable subroutines and/or functions in order to be compatible with the generic system. Also, care must be taken in the design of the procedures so that 1) the return path through the overlay structure is preserved, 2) procedures called by completion routines adhere to the restrictions for that type of subprogram, 3) procedures called as synchronous extensions to interrupt handlers do not cause the issuance of EMTs (emulator traps intructions) in addition to the above restrictions for completion routines, and 4) as an added restriction for completion routines or synchronous extension routines there may not be any overlay operations invoked, ie., all subprograms called must exist in the permanently resident program segment.

UCHECK

This function is called by FLDUPD when DTABLE(6,n) contains a non-zero value for the field being updated. The call is made immediately prior to the call to CVTFLD, which is normally used to convert and assign a value to the field being updated. A returned

value of zero indicates processing is complete except for the reporting of any detected errors (i.e. the field has been updated, refreshed on the screen, and FIELD has been reset if no field is to be addressed on the next pass through the control cycle) and a returned value of one indicates that CVTFLD will be called after reporting any detected errors.

UERROR

This subroutine is called by ERROR immediately after defining the diagnostic output line but before interpretation of the error code. Further interpretation of the error code into the normal set of messages is inhibited by setting the error code to zero. The error count in the user-performance data area will not be incremented except when recognizable error codes are interpreted into the normal set of messages.

UINITL

This subroutine is called by INITL after the initial screen erase, the assignment of the standard logical units and completion routines, and initialization of the user-performance data area.

ULOG

This subroutine is called by LOGON or LOGOFF (as indicated by a flag) immediately before resetting the user-performance data area. In both cases the screen has been erased, the system is in the no-record-yet-accessed condition, and the control record is current. When the calling subroutine is LOGON, the touch panel is active.

The touch panel has already been deactivated when the calling subroutine is LOGOFF.

UMSG

This subroutine is called by MESSAG immediately after defining the diagnostic output line but before interpretation of the message code. Further interpretaion of the message code into the normal text is inhibited by setting the message code to zero. A message code of minus one will immediately erase the diagnostic output line.

UPRMPT

This subroutine is called by (main) at the beginning of each control cycle after the echo line is defined but before the command read is initiated. The user-performance data area has been updated to reflect the most current command before the call is made.

USCAN

This function is called by SCAN immediately prior to the normal interpretation of the current command line by CMNDIN. A returned value of zero indicates that processing is complete except for the reporting of any detected errors, and a returned value of one indicates that CMNDIN should be called after reporting any detected errors. Only commands which are processed by CMNDIN will result in the updating of the user-performance data area.

UTEST

This function is called by RECTST in the evaluation of a Boolean "Select" request prior to the normal evaluation of an element of the relational expression table by COMPAR. A binary zero (false) or one (true) may be returned, or a minus one may be used to flag that normal evaluation by COMPAR is requested.

UTIME

This subroutine is called by subroutine TIME for both of the subroutines FDISPL and SDISPL, by FORTRAN IV completion routine TPANEL, or as a synchronous extension to the plasma panel interrupt server preceding each character generation. All calls are made immediately prior to an operation which will result in alphanumeric text being written to the plasma panel. If module NPRDCC is compiled with the /D switch, (i.e. UTIME is used as a synchronous extension to the plasma panel interrupt hardware), care must be taken that UTIME does not cause an EMT to be issued. Otherwise, the usual restrictions normally applying to completion routines must be observed unless the touch panel is disabled by ULOG. A flag passed to UTIME is used to indicate from where it was called.

UTOUCH

This subroutine is called from FORTRAN IV completion routine TPANEL before any other processing occurs in response to a touch panel interrupt. The restrictions applying to completion routines must be observed.

UWRAP

This subroutine is called from WRAPUP during program termination after interrupts to the plasma panel and touch panel have been disabled.

APPENDIX A

STATE TABLES

This appendix contains a tabular description of the finite state automaton used in the parsing of arbitrarily complex boolean search specifications by FORTAN IV subprogram BSCAN and the operator precedence push-down automaton used in the evaluation of the parsed expressions by assembly subprogram BOOLN.

The entries in both tables are constructed similarly, having the form:

A

S

R

where "A" represents the action or action(s) to be taken; "S" is the next state (if different than the current state); and "R" is a control variable value (for BSCAN) or statement label (for BOOLN) indicating which section of each subprogram is actually executed.

TABLE 1: State Table for BSCAN

CURRENT TOKEN

CURRENT STATE: State Number	Previous Token	0 ;	1 (	2 3 & |	4 ¬	5)	6 string	7 10 8 11 9 12 >=≥<≠≤
1	initial	-32	I 2 1	-8	4 3	-6	3 2	-7
2	(	-11	I 2 1	-9	4 3	-10	5 4	-12
3	string	7 T 9	-4	7 8 5	-14	7 D T 9 8	-14	-13
4	¬	-17	I 2 1	-15	4 3	-16	3 2	-13
5	(string	-20 8	-4	7 8 5	-14	7 D 9 8	-3	V 6 11
6	>=≥<≠≤	-23	-5	-22	-24	-21	C 7 12	-2
7	value	-33	-19	-25	-34	D 9 7	-26	-27
8	& |	-31	I 2 1	-29	4 3	-30	3 2	-28
9	)	T 10	-35	8 6	-38	D T 9 7	-37	-36

(Diagonal header labels: State Number; Previous Token; Current Token Type)

```
Actions:  I  =>  increment parenthesis counter
          D  =>  decrement parenthesis counter
          T  =>  if parenthesis counter not balanced, error <- -1
          V  =>  relational operator code <- current token value - 6
          7  =>  relational operator code <- 7
          C  =>  close relational expression

States:   #  =>  ISTAT <- #

References:  #  =>  DEST <- #
```

TABLE 2: State Table for BOOLN

Stack Number	Stack Top	CURRENT TOKEN: -1 (	-2 ε	-3 |	-4 ¬	-5)	0 ;	>0 operand
0	initial or (	A 1	-	-	A 2	-	-	E 1 2
1	operand	- 	A 2 3	A 4 4	-	D 1 5	B 6	-
2	ε	A 0 1	-	-	A 2	-	-	E 3 8
3	εoperand	- 	A 2 3	C 4 10	-	D 1 5	B 6	-
4	|	A 0 1	-	-	A 2	-	-	E 5 9
5	|operand	- 	A 2 3	C 4 10	-	D 1 5	B 6	-

```
Actions:  A  =>  push operator
          B  =>  evaluate stack to top; return result;
                 error if parentheses unbalanced
          C  =>  evaluate stack to top or (; push result
          D  =>  evaluate stack to (; push result;
                 error if parentheses unbalanced
          E  =>  evaluate unary operators; push result

States:   #

References:  1  =>  RECURS
             2  =>  PUSH
             3  =>  PUSH2
             4  =>  PUSH4
             5  =>  EVALR
             6  =>  EVALS
             7  =>  PEVAL1
             8  =>  PEVAL3
             9  =>  PEVAL5
            10  =>  EPUSH4
```

- finis -